APPENDIX A

PURPOSE

This Appendix A describes the XCS Algorithm and offers a scheme for adopting it to optimize the Digital Deal rules.

OVERVIEW OF CLASSIFIER SYSTEMS

A classifier system is a machine learning system that uses "if-then" rules, called classifiers, to react to and learn about its environment. Machine learning means that the behavior of the system improves over time, through interaction with the environment. The basic idea is that good behavior is positively reinforced and bad behavior is negatively reinforced. The population of classifiers represents the system's knowledge about the environment.

A classifier system generally has three parts: the performance system, the learning system and the rule discovery system. The performance system is responsible for reacting to the environment. When an input is received from the environment, the performance system searches the population of classifiers for a classifier whose "if" matches the input. When a match is found, the "then" of the matching classifier is returned to the environment. The environment performs the action indicated by the "then" and returns a scalar reward to the classifier system.

FIG. 7 generally illustrates one embodiment 700 of a classifier system.

One should note that the performance system is not adaptive; it just reacts to the environment. It is the job of the learning system to use the reward to reevaluate the usefulness of the matching classifier. Each classifier is assigned a strength that is a measure of how useful the classifier has been in the past. The system learns by modifying the measure of strength for each of its classifiers. When the environment sends a positive reward then the strength of the matching classifier is increased and vice versa.

This measure of strength is used for two purposes. When the system is presented with an input that matches more than one classifier in the population, the action of the classifier with the highest strength will be selected. The system has "learned" which classifiers are better. The other use of strength is employed by the classifier system's third part, the rule discovery system. If the system does not try

new actions on a regular basis then it will stagnate. The rule discovery system uses a simple genetic algorithm with the strength of the classifiers as the fitness function to select two classifiers to crossover and mutate to create two new and, hopefully, better classifiers. Classifiers with a higher strength have a higher probability of being selected for reproduction.

OVERVIEW OF XCS

XCS is a kind of classifier system. There are two major differences between XCS and traditional classifier systems:

1. As mentioned above, each classifier has a strength parameter that measures how useful the classifier has been in the past. In traditional classifier systems, this strength parameter is commonly referred to as the predicted payoff and is the reward that the classifier expects to receive if its action is executed. The predicted payoff is used to select classifiers to return actions to the environment and also to select classifiers for reproduction. In XCS, the predicted payoff is also used to select classifiers for returning actions but it is not used to select classifiers for reproduction. To select classifiers for reproduction and for deletion, XCS uses a fitness measure that is based on the accuracy of the classifier's predictions. The advantage to this scheme is that since classifiers can exist in different environmental niches that have different payoff levels and if we just use predicted payoff to select classifiers for reproduction then our population will be dominated by classifiers from the niche with the highest payoff giving an inaccurate mapping of the solution space.
2. The other difference is that traditional classifier systems run the genetic algorithm on the entire population while XCS uses a niche genetic algorithm. During the course of the XCS algorithm, subsets of classifiers are created. All classifiers in the subsets have conditions that match a given input. The genetic algorithm is run on these smaller subsets. In addition, the classifiers that are selected for mutation are mutated in such a way so that after mutation the condition still matches the input.

XCS CLASSIFIERS

A Classifier is an "if-then" rule composed of 3 parts: the "if", the "then" and some statistics. The "if" part of a classifier is called the condition and is represented by a ternary bitstring composed from the set {0, 1, #}. The "#" is called a Don't Care and can be matched to either a 1 or a 0. The "then" part of a classifier is called the action and is also a bitstring but it is composed from the set {0, 1}. There are a few more statistics (see table below) in addition to the Predicted Payoff and Fitness that were mentioned above.

Example of a Classifier:

0#011#01##000011#1 $\Rightarrow$ 011010

The condition (the left-side of the arrow) could translate to something like "If its Thursday or Tuesday at noon and the order is a Big Mac and Soda."
The action (the right-side of the arrow) could translate to something like "Offer an ice cream cone."

CLASSIFIER MATCHING

It was stated above that the population of classifiers is searched for classifiers that match the input. How does a classifier match an input? First, the input from the environment (like Big Mac and Coke) is encoded as a string of 0's and 1's. A classifier is said to match an input if: 1. The condition length and input length are equal 2. For every bit in the condition, the bit is either a # or it is the same as the corresponding bit in the input. For example, if the input is "Thursday, noon, Big Mac, Soda" then there might be a classifier that has a Don't Care for the day of the week. If there is such a classifier then it would match the input if it also has "noon, Big Mac, Soda" in the condition.

Example of Matching:
Let the input from the environment be:

I: 001010011 (Could mean something like: Thursday, 1:00 pm, Cashier 2, Store 10, 2 Big Macs, 1 Large Coke)

Let the population of classifiers be:

C1: 01##110## $\Rightarrow$ 0110

C2: #010#001# $\Rightarrow$ 1000

C3: 0#1#100## $\Rightarrow$ 0111

C4: 0#111#0#0 $\Rightarrow$ 0110

C5: 00#1000#0 $\Rightarrow$ 0010

C6: 0##0100## $\Rightarrow$ 0001

I matches C2, C3, C6.

CLASSIFIER STATISTICS

The following table 1 lists the statistics that each classifier keeps along with the algorithm for updating the statistics after a reward has been received from the environment.

STATISTIC	DESCRIPTION	UPDATE ALGORITHM Let L be the Learning Rate Let R be the Reward received The "If (experience < 1/L)" is the implementation of the MAM technique

Prediction	Keeps an average of the expected payoff if the classifier matches the input and its action is taken. **Note that fitness is used to select classifiers for reproduction only. Prediction is used to define which is the "best" classifier.**	If (experience <=1/L)
		pred = (pred * experience + R) / (experience + 1)
		Else
		pred = pred + L * (R – pred)

Error	Estimates the errors made in the prediction.	If (experience <= 1/L) error = (error * experience + (| R – pred | / paymentRange)) /

		(experience + 1) Else error = error + (L * ((| R – pred | / paymentRange) - error))
Fitness	The fitness of the classifier is based on the accuracy of the classifier's predictions. Note that fitness increases as error decreases. **Note that fitness is used to select classifiers for reproduction only. Prediction is used to define which is the "best" classifier.**	First, calculate the total accuracy for all classifiers in the action set. TotalAccuracy TA = $\square_{c \text{ in Action Set}}$ (numerosity$_c$ * Accuracy$_c$) Second, compute relative accuracy, RA. RA = (accuracy * numerosity) / TA. Then, compute fitness. fitness = fitness + L * (RA – fitness)
Experience	The number of times since its creation that a classifier has belonged to an action set.	Increment By 1
GA Iteration	Denotes the time-step of the last occurrence of a GA in an action set to which this classifier belonged.	Set to current iteration

Action Set Size	Estimates the average size of the action sets this classifier has belonged to. Updates to this are independent of updates to fitness, error and prediction.	If (experience <= 1/L)
		size = size +
		($\square_{c \text{ in Action Set}}$ numerosity$_c$ - size) /
		experience
		Else
		size = size +
		L * ($\square_{c \text{ in Action Set}}$ numerosity$_c$ – size)

Numerosity	Is the number of microclassifiers that are represented by this classifier.	Incremented when a classifier subsumes another classifier and when an identical classifier is created. Decremented when a classifier is deleted from the population. If numerosity equals 0 then the classifier is deleted from the population.

Accuracy	This is a measure of how accurate a classifier's predictions are. This can be computed from error so it does not need to be stored.	Let E be the minimum error
		If (error <= E)
		Accuracy = 1.0
		Else
		Accuracy =

		$e^{((\ln(\text{fallOffRate}) * (\text{error} - E)/E)} * \text{fallOffRate}$ Note: fallOffRate < 1 => ln(fallOffRate) < 0 error > E => error – E > 0 e raised to a negative power is a number in (0,1) so Accuracy becomes some number between (0,1)

TABLE 1

INPUT COVERING – GENERATION OF MATCHING CLASSIFIERS

When an input is received, the population of classifiers is searched and all matching classifiers are put in a set called the Condition Match Set. If the size of the Condition Match Set is less than some number N then the input is not covered. The number N is known, appropriately enough, as the Minimum Match Set Size and is a parameter of the system. To cover an input, matching classifiers are created and inserted into the population.

The algorithm for creating matching classifiers is as follows:

1. Initialize the classifier, CL, so that its condition identically matches the input.
2. For each bit in CL: Generate a random number, R, in [0,1]. If (R < Covering Probability) then change the bit to a '#'. Covering Probability is also a parameter of the system.
3. Generate a random action that is not present in the Condition Match Set.
4. Set the prediction equal to the mean prediction of all classifiers in the population.
5. Set the error equal to the mean error of all classifiers in the population.
6. Set the fitness equal to the 0.1 * mean fitness of all classifiers in the population.
7. Set the experience equal to 0
8. Set the GA iteration equal to the current iteration.
9. Set the action set size equal to the mean action set size.

10. Set the numerosity equal to 1
11. Insert CL into the population and into the Condition Match Set

DIGITAL DEAL CLASSIFIERS

Digital Deal classifiers are just like regular XCS classifiers except that they have special requirements for matching, covering and random action generation. Both the condition and action contain Menu Item Ids. These are used to look up the item in the Digital Deal menu item database in order to get pricing and cost information. The Digital Deal classifiers are stored in the DPUM database.

CONDITION

The condition in a Digital Deal classifier is 3 64 bit chunks for the environment and 6 128-bit chunks for the food items. The environment contains things like day-of-week, time-of-day, cashier id, store id, etc. Calling the right-most bit the 0^{th} bit, the following table 2A defines the bit locations of each field in the environment:

Bits	**Field**	**Len**
0 – 32	Destination ID from DPUM database	33*
33 – 44	Month (Jan => 1, Feb => 2, Mar=>4, etc) of Order	12
45 – 49	Time of Order – Hour	5
64 – 96	Period ID from DPUM database	33*
97 – 103	Day Of Week (Sunday => 1, Monday => 2, Tuesday => 4, etc)	7
128 – 159	Register ID from DPUM database	32
160 – 191	Cashier ID from DPUM database	32

* MSB is the sign bit, if set then the quantity in the remaining bits is negative

TABLE 2A

Each of the next 6 128-bit chunks defines a menu item. Calling the right-most bit the 0^{th} bit, the following chart defines the bit locations of each property of a menu item:

Bits	Property Name	Len
0 – 11	Menu Item Type	12
12 – 23	Size	12
24 – 35	Temperature	12
36	Pre-packaged	1
37	Discounted	1
38 – 43	Time Of Day Available	6
64-127	Specific Properties for Type	64

The exact values for the Property Name column are defined in Appendix A-2.

TABLE 2B

ACTION

An action has a variable length. The length depends on the type of action and the length of the binary descriptions of the menu items in the action. The shortest possible length of an action is 3 * 64 bits and the length will always be a multiple of 3.

An action is composed of groups of 3 64-bit chunks. The first chunk contains the 32-bit Menu Item Id from the DPUM database and the next 128-bits contain the binary description of that menu item. If the item is a meal then it will need more than one 128-bit chunk for the description so append the additional 128-bit description with a pad of 64 0's between each 128-bit description.

If the action is a Replace then the first Menu Item Id is the Id of the item to replace and the second Menu Item Id is the Id of the offer. If the action is an Add then there will only be one Menu Item Id in the action. Additionally, the MSB of the first 64-bit chunk will be set if the action is a Replace.

DIGITAL DEAL CLASSIFIER MATCHING

Before an order is sent to the XCS system, it is broken up into separate meals. Exactly how the order is broken up is discussed later but here is an example: Let the order be 1 Big Mac, 1 Hamburger, 2 Large Fries, 1 Coke, 1 Apple Pie then the possible meals are M1 = (Big Mac, Large Fries, Coke, null, null, null) and M2 = (Hamburger, Large Fries, Apple Pie, null, null, null). A meal contains 6 menu items. Some of the menu items may by null. A menu item belongs to one of 6 classes: main, side, beverage, dessert, miscellaneous, topping/condiment. A meal may have more than one kind of menu item in it (e.g., it is ok for a meal to have 2 sides). The input that we are matching against is actually a meal and not an entire order.

With all of that in mind, for a classifier, C, to match a given input, I, then all of the following must be true:

1. The environments of I and C must match. The first 192 bits of C and of I are the environment. Use traditional bit-by-bit matching to match the two environments .
2. Use traditional bit-by-bit matching to match the menu items. For each menu item in the input, there must be a matching menu item in the classifier. Order does not matter. The first item in the input can match, say, the third item in the classifier.
3. The action must match the input. For example, if the input is "Big Mac and Soda" then the action cannot be "Replace the small coffee with a large coffee."
4. The amount of change must be less than the price of the offer. For example, if the total price of the order is $2.01 then the change is $0.99 and if the price of the offer in the action is $0.50 then this is not a match. This classifier could have been created for an order with a total price of something like $2.60 so that the action with a price of $.50 made more sense.

DIGITAL DEAL RANDOM ACTION GENERATION

The process of generating random Digital Deal actions may seem like a trivial task but is quite complicated. The chief culprit is the desire for the random actions to be very random. By "very" random, I mean that the search space of all possible actions is quite large so the random actions should cover as much of it as possible. The other major problem is that the random actions are subject to a whole slew of constraints. The actions generated should be profitable to both the store and the customer. For example, an offer that is not profitable to the store is "For your change of $0.05, add 20 Big Macs" and an offer that is not profitable to the customer is "For your change of $0.30, you can replace your Super-Size soda with a small Soda." Remember that the order is broken up into meals so random actions are generated per meal.

The following is a step-by-step explanation of how random actions can be generated.

1. Let TP be the total price of the entire order (not just the meal).
2. Let T be the time of day that the offer is valid (e.g., the Period ID of the order).
3. Initialize O, the set of possible offers, to the empty set.
4. With equal probability, randomly decide if the offer will be a replace or an add.
5. If the offer is a replace then randomly pick something from the meal to replace. The item can be replaced if it's parent item is null and it's min and max price are > 0.
6. Let TP_{round} be TP rounded up to the next dollar.
7. Compute the amount of change available by subtracting TP from TP_{round}.
8. If the offer is an add then add all menu items that satisfy the following to O: the item is for the presently described embodiment of the invention, the min price is less than the change, the max price is greater than the change and the item is available in time period T. If the offer is a replace then add all menu items that satisfy the following to O: the item is for the presently

described embodiment of the invention, the price of the item is greater than the price of the replaced item, the (min price – min price of replaced) is less than the change, the (max price - max price of replaced) is greater than the change and the item is available in time period T. For a replace, we have to check both price and max price since the max price of an item may be 0 if it is not available as an offer.

9. If the size of the set O generated in Step 8 is less than half the size of the minimum match set size (M) then add $1 to the change and return to Step 8 to try to add more items to O. By making the size of the offer pool greater than M, as opposed to just greater than 0, we are guaranteed to have more random actions.
10. If the set O is not empty then randomly select one of the items and return it. If the set is empty and the offer is a replace then switch the offer to an add and go to step 8. If the set is empty and the offer is an add then return null; no offer will be generated for this order.

XCS SYSTEM PARAMETERS

The following TABLE 3 lists the system parameters for the XCS algorithm. An application with a graphical interface may be built to allow an expert user to change these parameters. The given defaults are the defaults recommended by the designer of the XCS algorithm (see Wilson 1995 referenced above).

PARAMETER	DESCRIPTION	COMMON SETTING	DEFAULT
Population Size	Number of classifiers in the system	This should be large enough so that covering only occurs at the very beginning of a run.	5000
Action Space Size	The number of possible actions in the system.	It must be greater than the minimum match set size.	85
Initial Prediction	The initial classifier prediction value used when a classifier is created through covering.	Very small in proportion to the maximum reward. For a maximum reward of 1000, a good value for this is 10.	10
Initial Fitness	The initial classifier fitness value used when a classifier is created	0.01	0.01

	through covering.		
Initial Accuracy	The initial classifier accuracy value used when a classifier is created through covering.	0.01	0.01
Initial Error	The initial classifier error value used when a classifier is created through covering.	Should be small	0
Crossover Probability	The probability of crossover within the GA	Range of 0.5 - 1.0	0.8
Mutation Probability	The likelihood of a bit being mutated	Range of 0.01 – 0.05	0.04
Minimum Match Set Size	The minimal number of classifiers in the match set that must be present or covering will take place	To cause covering to provide classifiers for every action then set this equal to the number of available actions.	10
GA Threshold	The GA is applied in a set when the average time since the last GA is greater than this threshold. Each classifier keeps track of a time stamp that indicates the last time that a GA was run on an action set that it belonged to. The time stamp is in units of "steps."	Range 25 – 50	25
Covering Probability	The probability of using a '#' symbol in a bit during covering.	0.33	0.33
Learning Rate	The learning rate for Prediction, Error and Fitness. Used to implement the MAM technique.	0.1 – 0.2	0.2
Deletion Threshold	If the experience of a classifier is greater than this then the fitness of the classifier may be considered in its probability of deletion.	20	20
Exploration Probability	The probability that during action selection the action will be chosen randomly.	0.5	0.5
Minimum	The error below which	0.01	0.01

Error	classifiers are considered to have equal accuracy. Used to update the fitness.		
Fall Off Rate	Used to update the accuracy	0.1	0.1
Subsumption Threshold	The experience of a classifier must be greater than this in order to be able to subsume another classifier.	20	20
Mean Fitness Fraction	Specifies the mean fitness in the population below which the fitness of a classifier may be considered in its probability of deletion.	0.1	0.1
Minimum Reward	The reward for a bad action.	0	0
Maximum Reward	The reward for a good action.	1000	1000
Action Set Subsumption Flag	Action Set Subsumption can be turned on/off by toggling this flag.	True	True
GA Subsumption Flag	GA Subsumption can be turned on/off by toggling this flag.	True	True

TABLE 3

SINGLE-STEP XCS ALGORITHM

1. Let O be the order (For example, 1 KFC Meal (Chicken Leg, Cole Slaw, Beans), 1 Chicken Sandwich, 1 Soda, and an Apple Pie). Let C be the population of classifiers.
2. Break O into meals M_1, M_2, M_3, … M_N
    a. Shuffle the order of the items in the order
    b. For each item in the order, find the item in the Menu Item table. If the item cannot be found and the item's parent is null then reject the entire order and return no offer. If the item cannot be found but it's parent is non-null then just skip the item. If the item is of type Meal

(like a Extra Value Meal) then add it to a unique M_i. If the item is not of type Meal then place it into a separate list. After all the items in the order have been inspected, scroll through the list of single type items and add those to the recently created M_i or create new M_i.

For the example order above the possible meals are:

M_1 = Chicken Leg, Cole Slaw, Beans, Apple Pie, null, null

M_2 = Chicken Sandwich, Soda, null, null, null

3. For each Meal in the order, generate Condition Match Sets. Create a Condition Match Set by searching through the population for any classifiers that match the given Meal.
4. If the size of any Condition Match Set is less than the Minimum Match Set Size then cover the Meal. See the sections on Classifiers and Digital Deal Classifiers for an explanation of covering.
5. For all the Condition Match Sets, create a Prediction Array. The Prediction Array stores the predicted payoff for each possible action in the system. The predicted payoff is a fitness-weighted average of the predictions of all classifiers in the Condition Match Set that advocate the action. The formula for calculating the fitness-weighted averages is: Let AS be the set of classifiers from the Condition Match Set with the same action, A. Then the Predicted Payoff, P, of A is: $P = (\sum_{c \in AS} \text{Prediction}_c * \text{Fitness}_c) / \sum_{c \in AS} \text{Fitness}_c$
6. If possible, choose 2 actions. The actions can be either a random selection (exploration) or based upon the Prediction Array (exploitation). If exploration then choose 2 random actions. If exploitation then choose the 2 best actions. The best action is defined to be the action with the highest prediction. If the highest prediction is shared by two or more actions then randomly choose an action.
7. Create an Action Set for each chosen action. The Action Set is the set of classifiers from the Condition Match Set that have actions that match the chosen action. The Genetic Algorithm is run only on the Action Set.

8. Return the actions to the environment. The amount of the reward is based on whether the offer was rejected or accepted. The reward is 0 if the offer was rejected. If the offer was accepted then the amount of the award is (1 – minPrice of offer/change in order) * 100 rounded to the nearest integer and then divided by 10. This gives rewards in the set {1000, 1100, 1200, ..., 2000}. This reward scheme gives accepted offers with bigger profits a higher reward. Since two offers are returned, the accepted offer is given a positive reward while the other offer is given a negative reward.
9. Using the reward, update all the statistics of the classifiers that are part of Action Set. The statistics are modified in the following order: experience, action set size prediction, error, accuracy and fitness. Changing the order of the modifications will change the rate at which the system learns. For example, if prediction comes before error then the prediction of a classifier in its very first update immediately predicts the correct payoff and consequently the prediction error is set to 0. This can lead to faster learning in simple processes but can be misleading in more complex problems. The algorithms for updating the statistics are given in a table above.Do Action Set Subsumption if it is enabled. In Action Set Subsumption, the Action Set is searched for the most general classifier that is both accurate and sufficiently experienced. All other classifiers in the set are tested against this general one to see if it subsumes them. Any classifiers that are subsumed are removed from the population. Example: Let the Action Set be:C1: 011#110## → 0111 C2: #010#001# → 0111 C3: 0#1#1#0## → 0111 C4: 0#111#0#0 → 0111. C3 is the most general since it has the most #'s. It is more general than C1 and C4. It is not more general than C2 since C2 has a '#' in the first position and C3 does not. If C3 is accurate and sufficiently experienced then we could subsume C1 & C4 by removing them from the population and increasing the numerosity of C3 by 2.
11. Run the Genetic Algorithm (GA) if the Action Set indicates that we should. The GA will be run on the Action Set if the average time since the last GA in the set is greater than the GA threshold. Average time, AT, is computed as follows:

$AT = \square$ GA iteration$_{cl}$ * numerosity$_{cl}$ $\square$ numerosity$_{cl}$) where the $\square$ is over the Action Set. To run the GA, use Roulette Wheel Selection to select two parents from the Action Set. By using Roulette Wheel selection, the classifiers with the highest accuracy tend to reproduce most often. Using the probability of crossover, the parents are crossed. If the parents are crossed then the prediction values of the offspring are set to the average of the prediction values of the parents. Notice that crossover only takes place in the condition and not in the action. Next, mutate the two offspring. Mutation takes place in both the action and the condition. XCS uses a restricted version of mutation that only allows a bit of the condition to be mutated if it is changed to a '#' or to a value that matches the given input. This results in an offspring with a condition that still matches the input. Actions are mutated as a whole (e.g., actions are mutated into a randomly generated new action).

Now that we have two new offspring, check if its parent subsumes either offspring. The parent must have an experience level greater than the Subsumption Threshold and must be accurate (accuracy of 1.0). If the offspring is subsumed then do not insert it into the population, just increment the numerosity of the parent. If the offspring is not subsumed then it is inserted to the population. If the size of the population is greater than the maximum size then a classifier has to be selected for deletion. XCS uses Roulette Wheel Selection to select a classifier for deletion.

ORGANIZATION OF THE SOFTWARE

The code is organized into two parts: the Classifier System and Digital Deal Classifier. The Classifier System is a black box that receives a vector of bitstrings, runs the XCS algorithm on them, produces an action and receives rewards. It knows nothing about Digital Deal, QSR, Big Macs, upsells, etc. The Classifier System contains an abstract object called Classifier. When the Classifier System is created, it is passed the name of a classifier class. This classifier class encapsulates all of the peculiarities of the problem at hand. Through the power of inheritance,

the Classifier System black box can manipulate Digital Deal classifiers or any other kind of classifier. The Digital Deal Classifier module supplies all the special routines for matching and generating random actions that were discussed above.

CLASSIFIER SYSTEM

SystemParameters

Each environment must create a SystemParameters class using the function *SystemParameters.createSystemParameters*. This function verifies that the parameters are valid and then creates and returns a reference to a SystemParameters class. If the parameters are invalid then an exception is thrown. This function takes a String argument. If the argument is null then the default system parameters are used. If the argument is not null then it must be the name of a SystemParameters class. A reference to the parameters class is passed to the ClassifierSystem when it is created. To change the defaults:

1. Derive a SystemParameters class from SystemParameters. Implement the function *localDefaultValues* to add new defaults values.
2. Pass the name of this new class to the function *SystemParameters.createSystemParameters*.

Additional parameters can be added in a similar way.

BitString

A BitString is a class containing an array of longs. In Java, longs are 64-bits long. When a BitString is created with just a length then:

1. Figure out how many 64-bit chunks are needed to contain that length. Example if length=65 then 2 64-bit chunks are needed.
2. Initialize the array of longs to have a length equal to the number of chunks that was computed in 1.
3. Initialize each element of the array to 0.

When a BitString is created with a String argument then:

1. Do the same as above using length = string length.
2. If the i-th character of the string is a '1' then figure out which bit in which chunk maps to i and set it to a 1. The mapping is from 1-Dimension to 2-Dimensions and is given in TABLE 4 below.

String Index	Array Index	Bit of Long
0	0	0
1	0	1
63	0	63
64	1	0
127	1	63
128	2	0
i	i / 64	i mod 64

TABLE 4

Each classifier is composed of two BitStrings, the condition and the action. The BitString class provides functions for creating BitStrings, for testing if two BitStrings are equal, for cloning a BitString, for accessing bits from a BitString and for modifying the bits of a BitString.

<u>ConditionBitString</u>

The ConditionBitString class is derived from the BitString class. This class has an additional array of longs which functions as a Don't Care mask. If any bit in the Don't Care mask is set then the corresponding bit in the original array is a Don't Care bit. The ConditionBitString class provides functions for determining if two ConditionBitStrings match. Using a series of exclusive-or operations tests matching.

<u>Classifier</u>

A Classifier is an abstract class. In order the use the XCS package, one must derive a Classifier class from this parent. Implementations for the functions *localInit* and *clone* must be provided. When the ClassifierSystem is created, it is given the name of the derived Classifier class so that any Classifiers that are created in the ClassifierSystem will be of the derived type.

A Classifier has three parts: a condition, an action and some statistics. Both the condition and action are BitStrings. A Classifier has two constructors: the public constructor is used to create a Classifier with an empty condition and empty action. The function *fillClassifier* must be used to actually set the condition and action.

The private constructor is only used to clone an existing Classifier. Functions are provided to mutate, crossover, test for equality, test for matching, modify the statistics, and read the statistics.

<u>ClassifierStatistics</u>

The ClassifierStatistics class encapsulates all of the classifier statistics. Functions are provided for accessing and modifying the statistics. The algorithms for updating the statistics are described in detail in the table found in the XCS Classifier Statistics section.

<u>ClassifierSystem</u>

The only interface with the outside world is through the ClassifierSystem class. One can create a ClassifierSystem, give an input to the system, receive an output from the system, give a reward to the system and query the system for the current classifier population. When a ClassifierSystem is created, it is given the name of the Classifier class to use when creating new classifiers and is given the system parameters to use in the execution of the XCS algorithm.

<u>ClassifierPopulation</u>

The ClassifierPopulation class contains the collection of classifiers that the XCS algorithm uses. Functions exist for inserting and deleting classifiers and for searching the population for classifiers that match an input.

<u>ConditionMatchSet</u>

The ConditionMatchSet class is used to create Condition Match Sets. A Condition Match Set is a collection of classifiers from the population whose condition matches a given input string. For traditional XCS classifiers, a classifier is said to "match" an input string if: 1. Condition length and input length are equal 2. For every bit in the condition, the bit is either a # or it is the same as the corresponding bit in the input. Matching for Digital Deal classifiers is much more complicated. A Condition Match Set is said to "cover" an input if the number of classifiers in the match set is at least equal to some minimum number. Functions exist for creating the prediction array from the match set, for enumerating the match set and to test if the match set covers an input.

<u>PredictionArray</u>

The prediction array stores the predicted payoff for each possible action in the system. The predicted payoff is a fitness-weighted average of the predictions of all classifiers in the condition match set that advocate the action. If no classifiers in the match set advocate the action then the prediction is NULL. Ideally, the prediction array is an array with a spot for each possible action. For our system, the number of possible actions is too big so we will only add actions for which a classifier advocating that action exists. Functions exist for creating a PredictionArray from a ConditionMatchSet, for returning the best action based on predicted payoff and for returning a random action. The fitness-weighted average is computed as follows:

1. For a given action, compute the weighted prediction. The weighted prediction is the sum of the prediction * fitness for each classifier advocating that action.
2. For a given action, compute the total fitness. The total fitness is the sum of the fitness for each classifier advocating that action.
3. The fitness-weighted average for an action is the weighted prediction / total fitness.

<u>ActionSet</u>

During the course of the XCS algorithm, an action is selected from all the possible actions specified in the Condition Match Sets. The ActionSet class contains the set of classifiers from the Condition Match Set that have actions that match the selected action. The GA is run only on the ActionSet. For each iteration of the XCS algorithm, a new ActionSet is formed. If the size of the Action Set is greater than one then action set subsumption takes place. In action set subsumption, the Action Set is searched for the most general classifier that is both accurate and sufficiently experienced. If such a classifier is found then all the other classifiers in the set are tested against this general one to see if it subsumes them. Any classifiers that are subsumed are removed from the population. Setting the subsumption flag in the system parameters to false can disable action set subsumption. Since the GA is run on the Action Set, it is not obvious how this algorithm can be used with historical data. Functions are included for updating all

of the classifier statistics, doing action set subsumption, and running the genetic algorithm.

XCSexception

This class is the exception class for the XCS algorithm. This exception is thrown when functions to implement the XCS algorithm are used incorrectly. For example, an XCSexception is thrown if one attempts to update the prediction before updating the experience.

DIGITAL DEAL CLASSIFIER

The Digital DealClassifier class is derived from the abstract class Classifier. As stated earlier, Digital Deal classifiers have special requirements for generating matching classifiers, generating random actions and checking for matching classifiers. This class provides all of the special functionality. When the ClassifierSystem is created then pass the name of this class to it.

INITIAL DIGITAL DEAL CLASSIFIER POPULATION

Since XCS is capable of generating classifiers, it can start with an empty population. However, the learning process is much quicker if XCS is given some knowledge with which to start. Since Digital Deal works well, it seems logical to seed the classifier population with the Digital Deal rules. The Initial Rule Generator application extracts the Digital Deal rules from the historical order and offer data. The application can be run from the Start Menu by choosing DPUM>BioNET Initial Rule Generator.

The BioNET.properties file is a flat property file that is used to configure the behavior of the application. The properties file can be found in c:\Program Files\DRS\DPUM\BioNET and can be edited with any editor. An explanation of the fields in the property file is given later.

ALGORITHM DESIGN

The following is a step-by-step explanation of the extraction and translation process.

1. Create the following tables in the database: The ClassifierCondition table has fields: Condition, Don't Care, Action Type, Experience, Action Set Size, Prediction, Fitness, Numerosity, Accuracy, Error, GA Iteration, The ClassifierAction table has fields for the action.

The ConditionAction table is the link table to link the condition and action.

2. Perform the following query to extract the orders from the order table: *SELECT OrderTable.OrderID, OfferItem.Replace, OrderTable.DestinationID, OrderTable.PeriodID, OrderTable.RegisterID, OrderTable.CashierID, OrderTable.DTStamp, OrderTable.Total, OrderItem.MenuItemID, OrderItem.Price, OrderItem.Quantity, OfferItem.MenuItemID, OfferItem.Quantity, OfferItem.OfferPrice, OrderItem.DPUMItem, OrderItem.ParentItemID, OfferItem.ReplaceMenuItemID FROM (OrderItem INNER JOIN OrderTable ON OrderItem.OrderID = OrderTable.OrderID) INNER JOIN OfferItem ON OrderTable.OrderID = OfferItem.OrderID WHERE (((OrderTable.OrderStatusID)=4) AND ((OfferItem.AcceptStatusID)=1) AND ((OrderItem.Deleted)=0)) AND (OrderTable.DTStamp IS NOT NULL) ORDER BY OrderTable.DTStamp DESC*
3. Using the first 10000 rows of the query result set, create QSRorder objects from all rows with the same Order ID.
4. Translate each QSRorder into 1 or more classifiers.
5. Add each classifier to a classifier population
6. For each classifier in the population, add Don't Cares to the condition.
7. For each classifier in the population, set the statistics to the default values.
8. Write the classifier population to the database.

<u>MODIFYING THE RUN-TIME BEHAVIOR OF THE INITIAL RULE GENERATOR</u>

The InitialRules application has a property file that is used to modify its run-time behavior. The following TABLE 5 is an explanation of the properties in the file.

Property Name	Description	Example
jdbc.drivers	Contains a list of class names for the database drivers. We are using the jdbc-odbc bridge so what is shown in the example is always valid.	sun.jdbc.odbc.JdbcOdbcDriver
jdbc.url	URL of the database to connect to. Since we are using the JDBC-ODBC bridge, the URL will start with "jdbc:odbc" and the last part must be set with the ODBC Data Sources tool in the Control Panel.	jdbc:odbc:McDs
jdbc.username	Login ID of the user to log into the database	sa
jdbc.password	Password needed to log the user into the database	
closedOrderStatusId	Value in the OrderStatusID column of the OrderTable table that indicates a closed order.	4
acceptStatusId	Value in the AcceptStatusID column of the OfferItem table that indicates an accepted offer.	1
numerosityMin	The minimum number of duplicates needed for a rule generated from an order to be written to the database. For example, if set to a 1 then every order will be translated to a rule and written to the database. If set to a 2 then the order must appear at least twice.	4
printClassifiers	Set to a 1 if you want the rules written to standard output as they are written	0

	to the database. Set to 0 otherwise.	
printOrders	Set to a 1 if you want the orders written to standard output as they are found. Set to a 0 otherwise.	0

TABLE 5

Properties are entered into the property file by typing propertyName=value. There should be no spaces between the name, =, and value. Notice that when a path and file name is given, the path can use forward slashes (/) or backward slashes (\) but when backward slashes are used they must be doubled. Java is case-sensitive so be careful.

TRANSLATING DIGITAL DEAL CLASSIFIERS TO ENGLISH

Using the Translation application, Digital Deal classifiers can be translated to English. Each classifier is translated to a string with each field delimited with the delimiter of your choice. The translation can then be exported to Excel or any other spreadsheet.

The Translator translates the Digital Deal classifiers into 3 different forms: a paragraph form, a parsed one-line form and into English. By far, the English version is the most useful but the other two forms are good for debugging.

The paragraph form parses each field (day of week, casher id, etc) of the classifier onto a separate line. The following is an example of one classifier translated into paragraph form:

```
----------CONDITION---------------
----------ENVIRONMENT-------------
Day of Week: 10#0#00
Period ID: 000#####000#00000##00####000000#0
Month: 00000000100#
Time of Day - Hour: ##001
Cashier ID: 00#000000##0##000000000##0#####0
Register ID: 000#00000000000#00000##0#00001##
Destination ID: 0000###0#00#0#0###0##000000#0#0##
-------------ITEM 1-----------------
```

Type: 0000#00###00

Size: 000000000010

Time of Day Available: #00110

Discounted: 0

Prepackaged: 0

Temperature: ####000##001

Side: 0000##00##00000#0##0##0#0#0000000001##00000#00###00###00#00#0000

-------------ITEM 2-----------------

Type: 0000##0000##

Size: 0###000##000

Time of Day Available: 00#000

Discounted: 0

Prepackaged: #

Temperature: 0#000##00000

Empty-Item: ##00#0#000#0000#000###0#0#00#0000#0000##0000000#0##000#000#0#000

-------------ITEM 3-----------------

Type: 000000#00##0

Size: 000000###0#0

Time of Day Available: 000000

Discounted: #

Prepackaged: 0

Temperature: ##000#0000##

Empty-Item: 00000#0000000000000000#000000000###0000000###0##0#000#00#000####

-------------ITEM 4-----------------

Type: 00#00##0###0

Size: 0000000000##

Time of Day Available: #0##00

Discounted: 0

Prepackaged: 0

Temperature: 000#0####00#

Empty-Item: 0000000000#0#0#000##000000#000##000##00##0000#000000#00##0###00#

-------------ITEM 5-----------------

Type: 0##00##0##0#

Size: 00000#000#0#

Time of Day Available: 00#00#

Discounted: 0

Prepackaged: 0

Temperature: 0#0000000###

Empty-Item: 000#0#00#00000000##0#0000#00##00#0###000#000000##00#00#0#0#00#00

-------------ITEM 6-----------------

Type: 0#0#000000##

Size: #0##0000#0##

Time of Day Available: 0#0000

Discounted: 0

Prepackaged: 0

Temperature: 000#00000000

Empty-Item: #0000#0#000000000#0#00#####0#000#00#0000000#000#00#00#0##0000#00

----------ACTION------------------

Action-Type: REPLACE

-------REPLACED ITEM------------

-------------ITEM 1-----------------

Menu Item Id: 11

Type: 000000000100

Size: 000000000010

Time of Day Available: 000110

Discounted: 0

Prepackaged: 0

Temperature: 000000000001

Side: 0000000000000000000000000000000001000000000000000100000000000

-------REPLACED WITH------------

-------------ITEM 1-----------------

Menu Item Id: 110

Type: 000000000100

Size: 000000000100

Time of Day Available: 000110

Discounted: 0

Prepackaged: 0

Temperature: 000000000001

Side: 0000000000000000000000000000000001000000000000000100000000000

N: 5 P: 10.0000 E: 0.0000 A: 0.0100 F: 0.0100 EXP: 0.0000 AS: 1.0000 GA: 0.0000

Condition ID: 1

Action IDs: 1, 2

The one-line parsed form is slightly more useful than the paragraph form. It returns each classifier on one line with a delimiter of your choice between each field. The output can then be exported to Excel to see the bits representing each field. The menu item id, condition id and action id are shown in decimal and not in binary. The following is an example using a '!' as the delimiter:

```
Condition ID!Day of Week!Period ID!Month!Time of Day - Hour!Cashier ID!Register
ID!Destination ID!Type!Size!Time of Day Available!Discounted!Prepackaged!Temperature!Type-
Properties!Type!Size!Time of Day Available!Discounted!Prepackaged!Temperature!Type-
Properties!Type!Size!Time of Day Available!Discounted!Prepackaged!Temperature!Type-
Properties!Type!Size!Time of Day Available!Discounted!Prepackaged!Temperature!Type-
Properties!Type!Size!Time of Day Available!Discounted!Prepackaged!Temperature!Type-
Properties!Type!Size!Time of Day Available!Discounted!Prepackaged!Temperature!Type-
Properties!Action-Type!Action ID!Menu Item ID!Type!Size!Time of Day
Available!Discounted!Prepackaged!Temperature!Type-Properties!Action ID!Menu Item
ID!Type!Size!Time of Day Available!Discounted!Prepackaged!Temperature!Type-Properties
1!10#0#00!000#####000#00000##00####000000#0!000000000100#!##001!00#000000##0##0000
00000##0#####0!000#00000000000#00000##0#00001##!0000###0#00#0#0###0##000000#0#0#
#!0000#00###00!00000000010!#00110!0!0!####000##001!0000##00##00000#0##0##0#0#0000
000001##00000#00###00###00#00#0000!0000##0000##!0###000##000!00#000!0!#!0#000##000
00!##00#0#000#0000#000###0#0#00#0000#0000##0000000#0##000#000#0#000!000000#00##0!
000000###0#0!000000!#!0!##000#0000##!00000#0000000000000000#000000000###0000000##
#0##0#000#00#000####!00#00##0###0!000000000##!#0##00!0!0!000#0####00#!000000000#
0#0#000##000000#000##000##00##0000#000000#00##0###00#!0##00##0##0#!00000#000#0#!0
0#00#!0!0!0#0000000###!000#0#00#00000000##0#0000#00##00#0###000#000000##00#00#0#0
#00#00!0#0#000000##!#0##0000#0##!0#0000!0!0!000#00000000!#0000#0#000000000#0#00###
##0#000#00#0000000#000#00#00#0##0000#00!REPLACE!1!11!000000000100!00000000010!0
00110!0!0!00000000001!00000000000000000000000000000000001000000000000000100000
000000!2!110!000000000100!000000000100!000110!0!0!00000000001!000000000000000000
00000000000000100000000000000001000000000000
```

The third form translates each field of the classifier to English and separates the fields by a delimiter of your choice. A good choice is '!' since the period id field often has '&' in it and the menu item field often has '$' and ',' in it. A detailed explanation of this form is given in section 5.

HOW DO YOU USE IT?

The application can be run from the Start Menu by choosing DPUM>BioNET Translator.

The BioNET.properties file is a flat property file that is used to configure the behavior of the application. The properties file can be found in c:\Program Files\DRS\DPUM\BioNET. This file can be edited with an editor and contains the following properties in TABLE 6:

Property Name	Description	Example
jdbc.drivers	Contains a list of class names for the database drivers. We are using the jdbc-odbc bridge so what is shown in the example is always valid.	sun.jdbc.odbc.JdbcOdbcDriver
jdbc.url	URL of the database to connect to. Since we are using the JDBC-ODBC bridge, the URL will start with "jdbc:odbc" and the last part must be set with the ODBC Data Sources tool in the Control Panel.	jdbc:odbc:McDs
jdbc.username	Login ID of the user to log into the database	Sa
jdbc.password	Password needed to log the user into the database	
separator	The delimiter for the fields of the English translations.	!
translatorOutputFile	Name of the file that the translated classifiers should be written to. If this file does not exist, it will be created. If it does exist, it will be overwritten. If the value is left blank then the translations will be sent to standard output.	c:/Program Files/DRS/DPUM/BioNET/trans.txt

TABLE 6

Properties are entered into the property file by typing propertyName=value. There should be no spaces between the name, =, and value. Notice that when a path and file name is given, the path can use forward slashes (/) or backward slashes (\) but when backward slashes are used they must be doubled. Java is case-sensitive so be careful.

WHAT'S IN THE ENGLISH TRANSLATION?

Referring to TABLE 7, the English translation shows what values of each field the condition will match to and what the action will be if that classifier is selected.

Field Name	Values	Example
Condition ID	Gives the ConditionID from the ClassifierCondition table in the DPUM database	Condition ID = 12
Day of Week	Lists the days of the week that this classifier will match to	Monday or Saturday
Period ID	Gives the periods that this classifier will match to	Lunch & Dinner or Late Breakfast
Month	Lists the months of the year that this classifier will match to	Apr or July
Time of Day – Hour	Lists the hours of the day (24 hour clock) that this classifier will match to	3 or 5
Cashier ID	Lists the names and ids of the cashiers that this classifier will match to	Gore, Al (45) or Bush, George(9)
Register ID	Lists the registers and ids of the registers that this classifier will match to	Far-Left (8) or Register 9 (3)
Destination ID	Lists the destinations that this classifier will match to	Front Counter or Drive-up
Ordered Items	Lists the ordered items and ids that this classifier will match to. Each classifier contains up to 6 menu items so the matches for each	[Cajun(17)] or [] or [] or [] or [] or []

	menu item are placed in brackets.	
Action-Type	Add or Replace	ADD
Action ID	Gives the ActionID from the ClassifierAction table in the DPUM database	Action ID = 23
Replaced or Offered Items	If the action is a REPLACE then this lists the item from the order that will be replaced. If the action is an ADD then this is the item to offer.	
Action ID	If the action is a REPLACE then this is the ActionID from the ClassifierAction table in the DPUM database If the action is an ADD then this will be blank.	Action ID = 26
Offered Item	If the action is a REPLACE then this is the menu item to offer. If the action is an ADD then this will be blank.	

TABLE 7

REPORTS

In addition to the Translator, there is a Reporting application that gives a summary of the Classifiers in the DPUM database. The reporting application provides the following information:

1. Number of Classifiers in the database
2. Number of Classifiers with ADD actions
3. Number of Classifiers with REPLACE actions
4. Top 10 most popular classifiers

5. Top 10 most likely to be selected classifiers (a.k.a. classifiers with the highest predictions)
6. Score of the database

The application can be run from the Start Menu by choosing DPUM>BioNET Reports.
The BioNET.properties file is a flat property file that is used to configure the behavior of the application. The properties file can be found in c:\Program Files\DRS\DPUM\BioNET. This file can be edited with an editor and contains the following properties described in TABLE 8:

Property Name	Description	Example
jdbc.drivers	Contains a list of class names for the database drivers. We are using the jdbc-odbc bridge so what is shown in the example is always valid.	sun.jdbc.odbc.JdbcOdbcDriver
jdbc.url	URL of the database to connect to. Since we are using the JDBC-ODBC bridge, the URL will start with "jdbc:odbc" and the last part must be set with the ODBC Data Sources tool in the Control Panel.	jdbc:odbc:McDs
jdbc.username	Login ID of the user to log into the database	Sa
jdbc.password	Password needed to log the user into the database	
separator	The delimiter for the fields of the English translations.	!
reportsOutputFile	Name of the file that the report will be written to. If this file does not exist, it will be created. If it	c:/Program Files/DRS/DPUM/BioNET/reports.txt

	does exist, it will be overwritten. If the value is left blank then the translations will be sent to standard output.	

Described 8

INSTALLATION OF BIONET-XCS

The BioNET-XCS is installed by running the InstallShield executable that is provided. It installs the actual BioNET and the four tools (Translator, Initial Rules, Reports and MenuEditor) in the directory c:\Program Files\Drs\Dpum\BioNET. To use the BioNET via DPUM, you have to edit the BioNET.properties file. Properties are described in TABLE 9.

Property Name	Description	Example
jdbc.drivers	Contains a list of class names for the database drivers. We are using the jdbc-odbc bridge so what is shown in the example is always valid.	sun.jdbc.odbc.JdbcOdbcDriver
jdbc.url	URL of the database to connect to. Since we are using the JDBC-ODBC bridge, the URL will start with "jdbc:odbc" and the last part must be set with the ODBC Data Sources tool in the Control Panel.	jdbc:odbc:McDs
jdbc.username	Login ID of the user to log into the database	Sa
jdbc.password	Password needed to log the user into the database	
breakfast	The PeriodID from the Period table in the DPUM database that denotes breakfast. If there is more than one id for breakfast then list them all separated by commas.	1,3,10
lunch	The PeriodID from the Period table in the DPUM	2

	database that denotes lunch. If there is more than one id for lunch then list them all separated by commas.	
dinner	The PeriodID from the Period table in the DPUM database that denotes dinner. If there is more than one id for dinner then list them all separated by commas.	2
anyPeriod	The PeriodID from the Period table in the DPUM database that denotes any period. If there is more than one id for any period then list them all separated by commas.	-2
logEnable	Set to a 1 if you want the output of the XCS algorithm logged to a file. This should only be a 1 for debugging since logging makes things very slow.	1
logFileName	Name of the file to output XCS logging to. If the file exists then new log messages are appended to it. If it does not exist then it is created.	c:/Program Files/DRS/DPUM/BioNET/xcsLog.txt

TABLE 9

REFERENCES

One of ordinary skill in the art may refer to the following references for a description of XCS.

Kovacs, T. (1996), “Evolving Optimal Populations with XCS Classifier Systems”, MSc. Dissertation, Univ. of Birmingham, UK.

Wilson, S. W. (1995), "Classifier Fitness Based on Accuracy", *Evolutionary Computation*, 3 (2), MIT Press.

Wilson, S. W., Butz, M. V. (2000), “An Algorithmic Description of XCS”, IlliGAL Report No. 2000017, University of Illinois at Urbana-Champaign.

THIS PAGE BLANK (USPTO)

APPENDIX A-1– XCS SYSTEM PARAMETERS

Classifier	A bit string encoding of an "if-then" rule where each bit can be either a 0, 1 or #. The '#' indicates a "don't care" and can be matched to either a 1 or 0. The "if" part of the classifier is called the **condition** and the "then" part is called the **action**. The action cannot contain any '#' characters. The format for a classifier is usually something like: 00##100#1110### ⇒ 101
Classifier System	A machine learning system that uses "if-then"rules to react to its environment. A genetic algorithm is used to discover new rules for the environment.
XCS	A classifier system where the fitness of a classifier is based on the accuracy of the payoff prediction as opposed to being based on the prediction itself.
GA	Genetic Algorithm
Condition Match Set	The set of classifiers that match the given input from the environment (e.g.. an order of a Big Mac). For example, suppose a Big Mac is encoded as 10010 and the condition parts of the classifiers in the population are: a. #0010 b. ###00 c. 1##10 d. 10010 e. 1##00 f. 10#0# Then the match set consists of: a, c, d.
Cover an Input	The process of creating a classifier that matches an input. If the Condition Match Set is empty than generate a classifier by taking the input and randomly replacing some of the characters with #'s and then randomly generating an action that is not present in the Condition Match Set.
Exploration	Randomly choose an action from the Condition Match Set.
Exploitation	Choose the best (as defined by the prediction array) action from the Condition Match Set.
Action Set	The set of classifiers from the Condition Match Set whose action matches the action that was chosen with either exploration or exploitation.

Microclassifier	Same as a classifier
Macroclassifier	If a classifier is created that has the same condition and action as another classifier then the existing classifier is said to be a Macroclassifier. Instead of adding a second identical classifier to the population, the **Numerosity** of the original classifier is incremented by 1. A classifier can only be deleted if its Numerosity is 0. If a classifier is marked for deletion and has a Numerosity of greater than 0 then decrement the Numerosity. The total number of classifiers in a population is the sum of the numerosities of all the classifiers in the population.
Subsumption	Let A and B be two classifiers with the same action. If the set of inputs that A will match is a superset of the set of inputs that B will match then A subsumes B.
GA Subsumption	If an offspring classifier is logically subsumed by the condition of an accurate and sufficiently experienced parent then the offspring is not added to the population but instead the numerosity of the parent is incremented. GA Subsumption can be disabled.
Action Set Subsumption	This takes place in the action set. The action set is searched for the most general classifier that is both accurate and sufficiently experienced then all other classifiers in the set are tested against this general one to see if it subsumes them. Any classifiers that are subsumed are removed from the population. Action Set Subsumption can be disabled.

Roulette Wheel Selection	A method of selection where each classifier is conceptually given a slice of a circular roulette wheel. The slice is equal in area to the classifier's fitness. A classifier is selected by spinning the wheel. The algorithm is as follows:
	Let fitnessSum = sum of fitness values for all classifiers in the action set
	Let randomPoint = random number in [0,1] * fitnessSum
	Set fitnessSum = 0
	For each classifier in the action set
	    fitnessSum = fitnessSum + fitness of classifier
	     if (fitnessSum > randomPoint)
	       Return (classifier)

Prediction Array	An array that stores the predicted payoff for each possible action in the system. The predicted payoff is a fitness-weighted average of the predictions of all classifiers in the Condition Match Set that advocate that action. If no classifiers in the Condition Match Set advocate that action then the prediction is NIL.
MAM Technique	Used to speed up the estimates of classifier parameters based on information obtained on successive cycles. Using this technique, a parameter is updated using one method early on and a second method later. The reasoning is that

	the first method can be used to quickly get a rough approximation of the true value, while the second method can make more conservative adjustments in order to refine the value.

TABLE 10

APPENDIX A-2 – FOOD ITEMS DATA MODEL

The general idea of the data model is to represent each item of an order by defining the item's properties. For example: Instead of saying a Big Mac is Menu Item #4, we will say that a Big Mac is something with Beef, Bread, Special Sauce, Lettuce, Tomato and a Pickle.

DESIGN GOALS

1. Design should be abstract enough to handle any food item from Extra Sour Cream at Taco Bell to Red Lobster's Shrimp Feast.
2. Design should introduce as little bias as possible.
3. Should be able to compare food items. This is the reason that numerical identifiers do not work. How does one compare a 5 to a 10? Numerical identifiers have no meaning. With an abstract model, we can talk about comparing the various properties of two items.
4. Should be able to compare food items from different brands. For example, compare Whoppers to Big Macs.

MODEL DESCRIPTION

An order is comprised of two objects: an Environment object and a Meal object.

ENVIRONMENT OBJECT

The Environment object consists of the following:

Time-of-Day

Destination (Take-out, Eat-in, Deliver, Drive-Thru)

Day-Of-Week

Payment Method

Customer ID

Store ID

Weather

Party Size

MEAL OBJECT

A Meal object consists of 6 Menu Item objects. Some of the Menu Item objects in a Meal can be NULL. There are 6 different kinds of Menu Item objects: Main, Side, Beverage, Dessert, Miscellaneous, Topping/Condiment. A Meal object does

not have to have one of each of the Menu Item types in it; it is perfectly valid for a Meal object to have, say, 2 Side Menu Items.

Examples of Meal objects:
Big Mac, Large Fries, Small Coke, NULL, NULL, NULL
Apple Pie, Coffee, NULL, NULL, NULL, NULL
Chicken Leg, Coleslaw, Baked Beans, Biscuit, Ice Cream, Iced Tea
Coke, NULL, NULL, NULL, NULL, NULL

<u>Menu Item Object</u>

A Menu Item comprises two things: an ID and list of binary-encoded properties. The ID is used only to query the Digital Deal database to get pricing and cost information and to get the name of the object to construct the offer string. Each Menu Item has a set of common properties and a set of properties that are unique to the Menu Item type. The properties are OR'ed together to form a binary descriptor. This descriptor must be stored in the Digital Deal database.

<u>Common Properties of a Menu Item</u>

Property Name	Value	Encoding
Type	Beverage Main Side Dessert Condiment Miscellaneous	000001 000010 000100 001000 010000 100000
Size If no size (like a Big Mac) is specified then the size is Medium.	Child Small Medium Large Extra-Large All-U-Can-Eat	000001 000010 000100 001000 010000 100000
Temperature	Hot	001

	Cold	010
	Room	100
Pre-packaged	False	0
	True	1
Discounted	False	0
	True	1
Time-Of-Day-Available	Any Time	111
	Breakfast	001
	Lunch	010
	Dinner	100

TABLE 11

Beverage Menu Item Properties

Property Name	Encoding
Water	0000000000000000000000000000000001
Milk	0000000000000000000000000000000010
Soda	0000000000000000000000000000000100
Fruit Juice	0000000000000000000000000000001000
Coffee	0000000000000000000000000000010000
Tea	0000000000000000000000000000100000
Beer	0000000000000000000000000001000000
Wine	0000000000000000000000000010000000
Liquor	0000000000000000000000000100000000
Chocolate	0000000000000000000000001000000000
Ice (like a Smoothie)	0000000000000000000000010000000000
Decaffeinated	0000000000000000000000100000000000
Diet	0000000000000000000001000000000000
Ice Cream	0000000000000000000010000000000000
Vegetable	0000000000000000000100000000000000
Protein-Shake	0000000000000000001000000000000000
Flavorings (like Vanilla, Orange, Fox's uBet Chocolate Syrup)	0000000000000000010000000000000000
Cappuccino	0000000000000000100000000000000000
Espresso	0000000000000001000000000000000000

TABLE 12

Main & Side Menu Item Properties

Name	Encoding
Egg	00000000000000000000000000000000001
Chicken	00000000000000000000000000000000010
Beef/Veal	00000000000000000000000000000000100
Lamb	00000000000000000000000000000001000
Turkey	00000000000000000000000000000010000
Pork	00000000000000000000000000000100000
Fish	00000000000000000000000000001000000
Seafood	00000000000000000000000000010000000
Other Meat	00000000000000000000000000100000000
Cheese	00000000000000000000000001000000000
Spices (Cajun, Blackened, Teriyaki, etc)	00000000000000000000000010000000000
Potato	00000000000000000000000100000000000
Onion	00000000000000000000001000000000000
Corn	00000000000000000000010000000000000
Mushroom	00000000000000000000100000000000000
Coleslaw	00000000000000000001000000000000000
Lettuce	00000000000000000010000000000000000
Peppers	00000000000000000100000000000000000
Other Vegetables	00000000000000001000000000000000000
Fruit	00000000000000010000000000000000000
Mayo	00000000000000100000000000000000000
Sauce/Dressing	00000000000001000000000000000000000
Soy (Tofu, Veggie-Burger, etc	00000000000010000000000000000000000
Nuts	00000000000100000000000000000000000
Beans	00000000001000000000000000000000000
Pasta	00000000010000000000000000000000000
Rice	00000000100000000000000000000000000

Is_Salad	00000010000000000000000000000000
Is_DeepFried	00000100000000000000000000000000
Is_Soup	00001000000000000000000000000000
Is_Sandwich (Taco, Burrito, Pita-Wrap etc)	00010000000000000000000000000000
Is_Pizza	00100000000000000000000000000000
Bread	01000000000000000000000000000000
Batter (Waffles, Pancakes)	10000000000000000000000000000000

TABLE 13

Dessert Menu Item Properties

Property Name	Encoding
Fruit	00000000000000000000000000000001
Pastry	00000000000000000000000000000010
Dairy (Cheese, Whipped Cream)	00000000000000000000000000000100
Chocolate	00000000000000000000000000001000
Cookie	00000000000000000000000000010000
Candy	00000000000000000000000000100000
Cake	00000000000000000000000001000000
Chips	00000000000000000000000010000000
Nuts	00000000000000000000000100000000
Coconut	00000000000000000000001000000000
Caramel	00000000000000000000010000000000
Is_CreamFilled	00000000000000000000100000000000
Is_FruitFilled	00000000000000000001000000000000
Frozen Treat	00000000000000000010000000000000
Batter	00000000000000000100000000000000
Ice Cream	00000000000000001000000000000000

TABLE 14

Miscellaneous Menu Item Properties

Property Name	Encoding

Toy	0000000000000000000000000000000001
Video	0000000000000000000000000000000010
Newspaper	0000000000000000000000000000000100
Salad Bar	0000000000000000000000000000001000

TABLE 15

<u>Topping/Condiment Menu Item Properties</u>

Property Name	Encoding
Salsa	0000000000000000000000000000000001
Cream Cheese	0000000000000000000000000000000010
Extra Dressing	0000000000000000000000000000000100
Sour Cream	0000000000000000000000000000001000
Butter	0000000000000000000000000000010000
Guacamole	0000000000000000000000000000100000
Fruit	0000000000000000000000000001000000
Dessert Topping (Sprinkles, Cookies, etc)	0000000000000000000000000010000000

TABLE 16

<u>Examples of Menu Item Encodings</u>

Regular McDonald's Apple Pie => Type = Dessert, Size = Medium, Temperature = Hot, Pre-packaged = True, Discounted = False, Time-Of-Day-Available = Anytime, Properties = Fruit, Pastry, Is_FruitFilled

Encoding = 00100 000100 001 1 0 111 0000000000000000000001000000000011

Senior Large Coke => Type = Beverage, Size = Large, Temperature = Cold, Pre-packaged = False, Discounted = True, Time-Of-Day-Available = Anytime, Properties = Soda

Encoding = 00001 001000 010 0 1 111 0000000000000000000000000000000100

CREATING BINARY DESCRIPTORS

We will need an application with a graphical interface to enter properties for menu items and categories.

The application may be something like the exemplary window 800 illustrated in FIG. 8:

Design considerations of the Menu Editor application:

1. Should be able to query the Digital Deal database for a list of the Menu Items and their properties.
2. Should be able to query the Digital Deal database for a list of the Categories and their properties.
3. Should be able to write the properties to the Digital Deal database.
4. Should be able to set the properties for a selected Menu Item or Category.
5. Should prevent the user from assigning dessert properties to a side item, etc.
6. Should have item templates like HAMBURGER, CHEESEBURGER, etc.